Log File Management Unification

(Version 1.2 - Hugo Pristauz, Eva Maria Sonnweber, Florian Speer)

*This document defines concepts and definitions for a unified machine log file management at the core level. It provides core standards which allow the implementation and usage of universal import drivers for analysis tools like MATLAB, as well as core standards for a log file and folder organization, comprising organization as projects and packages, including a standard naming convention for directory and file organization of log files.*

# 1 Introduction

# 1.1 Motivation

In order to provide machine diagnostics and to analyze and debug machine capabilities machine software is capable of generating several types of log files[[1]](#footnote-1). Efficient handling and processing of those log files requires a unified log file management which concerns both a standardization of log file formats and a standardization of file system organization. Here are some objectives which should be achieved with a unified log file management.

* supporting efficient analysis, diagnosis and debugging of machine capabilities (e.g., accuracy, temperature, force, TC process, …)
* providing an obligatory SW specification for log file management (every non-compliance is defined as a bug and has to be fixed)
* definition of a file organization, which allows automatic, semi-automatic and manual transfer of log file packages from machines to personal computers or data bases.
* MATLAB can be considered as the preferred analysis tool, thus the target is to have one universal log file import function (method) which can import any log file into MATLAB which complies to the specifications of the log file manage­ment system.

# 1.2 How To Get It Running

Log file management is concerning a huge field of applications and standardization should support all of these application areas. Thus the definition and implementation of an efficient log file management system finally ends up in definition and implementation tasks with tremendous efforts.

In order to move forward for such a big project a step-by-step approach concerning both definition and implementation is proposed. To support this kind of step-by-step approach the log file management system is suggested to be organized in terms of a layer structure.

# 1.3 Log File Management Structure

To enable a step-by-step approach for both definition and implementation of such log file management a bottom-up approach with layer organization is proposed (figure 1).

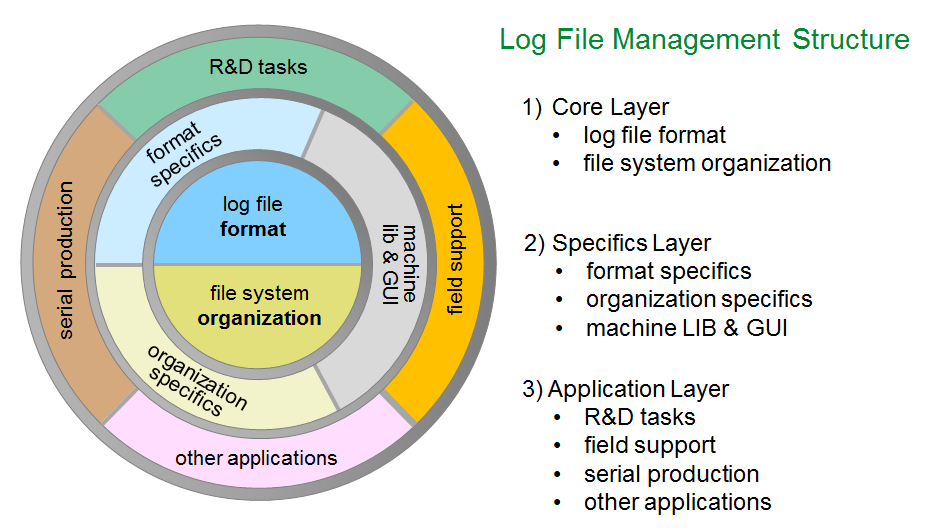


Figure 1: Log file management structure

# 1.4 Proposed Step-By-Step Approach

Based on the layer structure of the log file management the following step-by-step approach for definition and implementation is being proposed:

**Phase 1**

* Define a core log file management system comprising core format and core file organization
* Setup a log file management key group
* Align core definitions with key group
* First alpha tests

**Phase 2**

* Define log file format specifics with key group
* Each new logging has to be implemented according to core format and format specifics

**Phase 3**

* Define file organization with key group
* Implement log file management dialogs in machine SW (library)
* Each new logging has integrated log file dialogs

**Phase 4**

* Case-by-case backward integration of log file management
* Case-by-case implementation of data base functionality

# 2 Log File Format

# 2.1 What is Inside of a Log File?

Log files contain a bulk of information which can typically be categorized as follows (see figure 2).

* *type* information*:* tells us how *data* has to be interpreted
* *parameter* information: information which has a type-independent interpretation (like *title*, *comments*, *date*, *time*, *rows*, *columns, machine-ID, …)*
* *data* information*: all information which has a type specific interpretation*



defines how data

has to be interpreted

all information which has a

**type dependent** interpretation

all information which has a

**type independent** interpretation

Figure 2: three categories of log file information

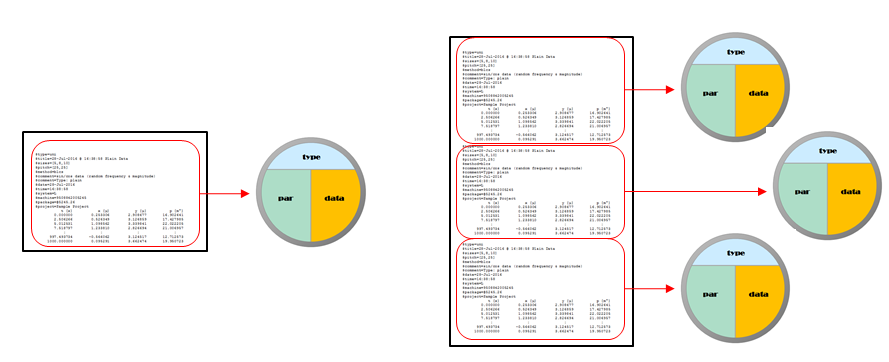
Beyond this elementary structure the log file may be organized into log file chunks which are called log objects. Each log object refers to *type*, *parameter* and *data* of a particular test (figure 3). At the core definition level there is some freedom whether a sequence of log file information chunks is logged into a single log file as a sequence of log file objects, or each part of the sequence is logged into a distinct log file representing only a single object

Figure 3: object organization of a log file

# Standard Log File Format

Log files should be represented in readable text file format and should contain all three types of information: *type*, *parameter* and *data*. A standard log file format is presented, which, by example may have the form of figure 2, figure 3 or figure 4[[2]](#footnote-2).

$type=uni

$title=28-Jul-2016 @ 16:38:58 Plain Data

$sizes=[5,8,10]

$pitch=[25,25]

$method=blcs

$comment=sin/cos data (random frequency & magnitude)

$comment=Type: UNI

$date=28-Jul-2016

$time=16:38:58

$system=L

$machine=9508842005245

$package=@5245.26

$project=Sample Project

$shape.kind='circle'

$shape.color.line='b'

$shape.color.fill='y'

$shape.width=[3]

$junk={[0,1.5],(struct('kind',[1],'color',(struct('line','b','fill','y'))))}

t [s] x [µ] y [µ] p [m°]

0.000000 0.253306 2.908677 16.902641

2.506266 0.526349 3.126859 17.427985

5.012531 1.098562 3.339841 22.022205

7.518797 1.233810 2.826694 21.006957

: : : :

997.493734 -0.566062 3.124517 12.712573

1000.000000 0.095291 3.662474 19.950723

Figure 2: Example of standard log file format

$type=uni

$title=28-Jul-2016 @ 16:38:58 Plain Data

$sizes=[5,8,10]

$pitch=[25,25]

$method=blcs

$comment=sin/cos data (random frequency & magnitude)

$comment=Type: UNI

$date=28-Jul-2016

$time=16:38:58

$system=L

$machine=9508842005245

$package=@5245.26

$project=Sample Project

$shape.kind='circle'

$shape.color.line='b'

$shape.color.fill='y'

$shape.width=[3]

$junk={[0,1.5],(struct('kind',[1],'color',(struct('line','b','fill','y'))))}

; t [s]; x [µ]; y [µ]; p [m°];

; 0.000000; 0.253306; 2.908677; 16.902641;

; 2.506266; 0.526349; 3.126859; 17.427985;

; 5.012531; 1.098562; 3.339841; 22.022205;

; 7.518797; 1.233810; 2.826694; 21.006957;

: : : :

; 997.493734; -0.566062; 3.124517; 12.712573;

; 1000.000000; 0.095291; 3.662474; 19.950723;

Figure 3: Example of universal log file format with delimiters

$type='uni'

$title='28-Jul-2016 @ 16:44:32 Simple Data'

$comment='sin/cos data (random frequency & magnitude)'

$comment='Type: UNI'

%

% these are three comment lines

%

$sizes=[6,10,8]

$pitch=[25,25]

$method='blrm'

$date='28-Jul-2016'

$time='16:44:32'

$system='L'

$machine='9508842005245'

$package='@5245.26'

$project='Sample Project'

$shape.kind='circle'

$shape.color.line='b'

$shape.color.fill='y'

$shape.width=[3]

$junk={[0,1.5],(struct('kind',[1],'color',(struct('line','b','fill','y'))))}

%

% the data header is a sequence of symbols followed by the unit in brackets

%

t [s] x [µ] y [µ] p [m°]

0.000000 9.880625 9.855523 81.474920

2.087683 9.942589 10.391461 85.133191

4.175365 10.625660 10.007634 89.020551

6.263048 10.734885 10.031975 90.672147

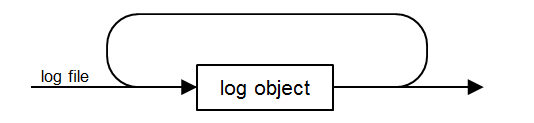
: : : :

997.912317 9.520618 10.462429 84.153708

1000.000000 10.098433 10.405004 88.089529

Figure 4: Example of universal log file format with alternative syntax

The log file syntax is given in figure 5. As displayed in the syntax diagram for *logfile* the sequence parameters-header-data can occur repeatedly in a single log file, the specific sequences would refer to specific objects.



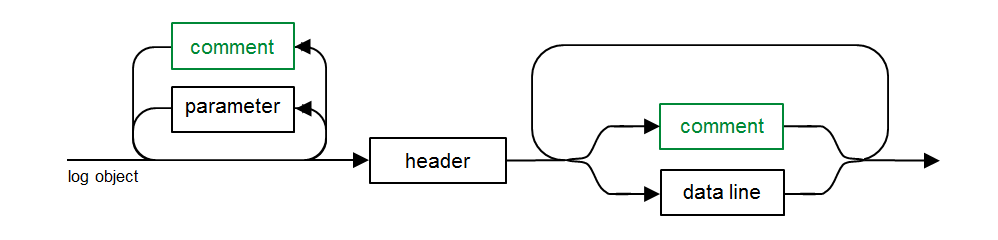


Figure 5: log file syntax

Here are some additional comments regarding the log file syntax.

* A parameter clause starts with an identifier complying to MATLAB syntax followed by the equal sign ('=') and ending with a value syntax.
* A value can be either string or numeric.
* String values follow either the MATLAB string literal syntax (like '5245.26', '28-Jul-2016') or follow a simplified string literal syntax which is without enclosing quotes. If a string value is with simplified syntax, it must not begin with a bracket ('[') or brace ('{').
* Numeric literals are always enclosed in brackets and supports any syntax which defines a matrix value (like: [3.14159] or [6,10,8] or [3 4 5; 7 8 9]).
* Cell array values are so far not supported, but might be extended in a straight forward way to be any valid MATLAB cell array literal to be enclosed between braces (like {'abc', 5, NaN; inf, pi, [5 6; 7 8]}).
* Any comment line starting with a percent character ('%') may be inserted in the parameter section of a log file

These are the most important topics to say about log file syntax. There are some additional comments to mention.

* A type declaration follows exactly the parameter clause syntax, however, the type declaration will be treated in a special way and not with the action of adding a type parameter during log file import.
* It is recommended to provide the type clause at the beginning of the log file, but providing the type clause at any line before the header is also valid.
* If parameter clauses are repeated (same parameter identifier in different lines) the parameters (except for the *comment* parameter) are 'overwritten', which means that the last occurring parameter clause will determine the value.
* An exception is the *comment* parameter. Multiple comment parameters are collected in a cell array, and will not overwrite previously found *comment* parameter clauses.
* the header line might be delimited with (any non-alpha-numeric) delimiter characters or might not (e.g. ';', '|'; ',' , …).
* If the header line has delimiter characters, the data lines must have the same delimiters.
* If the log file contains matrix based data (m x n = rows x columns) and the matrix is repeated r times then these dimensions should be provided by the *sizes* parameter (e.g. for 5 rows, 10 columns and 20 repeats sizes = [5,10,20]). The repeat parameter is optional (since it can be calculated by the import driver, e.g. $sizes=[5,10]), allthough it is recommended to provide also the repeat argument.
* For matrix based log data a *pitch* parameter clause should be provided which contains x/y pitch information of the matrix (e.g. pitch=[25,25])
* The processing method shall be provided as the *method* parameter. The meaning of the values are listed here by examples, where first character is 'b' or 't' (bottom or top), second character is 'l' or 'r' (left or right), third character is 'r' or 'c' (row or column wise), and fourth character is 's' or 'm' (saw-tooth or meander):

blrm start at bottom left, row wise by meander

brcs start at bottom right, row wise by saw tooth

tlrs start at top left, row wise by saw tooth

trcm start at top right, column wise by meander

* The *system* parameter should always be provided for multiple gantry or multiple bond head systems and shall have the value 'L', 'R' or 'D' (left, right or dual) for 8800 log files.
* The *machine* parameter (machine serial number) should be always provided (as string value)
* It is recommended to provide also a *project* and *package* parameter (string values). A *package* parameter should follow the syntax of figure 6. The *fourdigits*-part is comprised by the last for digits of the machine number, and *integer* is a continuously increased number (e.g. *@4567.34*, *@6720.235*, …)

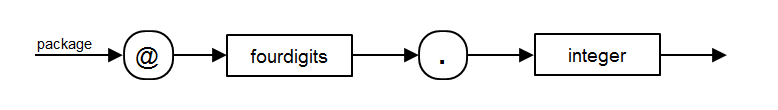


Figure 6: package identifier syntax

# File Organization

Any log file generated on a machine should be contained in a package folder, where the package folder naming should start with a package identifier syntax (see figure 6) followed by a period and type identifier, separated by a space or underscore or period character, followed by any sequence of valid file name characters. Examples are listed below:

@5245.26.VIB Vibration test substrate camera

@5245.27.VIB Vibration test upward camera

@5245.28.BMC BMC Test after calibration

If the package folder name includes a type identifier (like VIB or BMC) the folder should contain only log files with the specified type but no other types. This is not an absolute must but a strong recommendation to make it easier to find log files manually

The idea of the package identifier is to have a unique identifier for each log package which allows to identify a log file package uniquely, and to use the (compact) package identifier for labeling diagrams. Figure 7 shows a table of recommended type identifiers.

1. **Calibration & Quality Control**

BMC Basis Machine Capability Test

BQR Base Calibration & Quality Run

PUT Pull Test

SLS Slip Stick Test

VIB Vibration Test (Substrate or Upward Camera)

ABC Advanced Base Calibration

1. **R&D Test**

ANY Any Kind of R&D Test

DCT Distance Check Test

FRT Freeze Test

MBC Matrix BMC Test

PBI Post Bond Inspection Data

STL Stage Log Data

TPX Temperature/Position Crosstalk

1. **Thermocompression**

PAK Package Information File

TSK Task Information File

Figure 7: Recommended Type Identifiers

Package directories are organized by project directories and optional organizer directories, which might be nested. A project directory is always enclosed by brackets ('[' and ']').

# Default File Organization Structure

On a machine all packages are found inside the log root. The log root name is [log]. By default packages are stored as child folders of the log root, or as child folders of a project directory, which is a child folder of the log root (figure 8). The default file organization structure will be maintained default wise by the machine software.

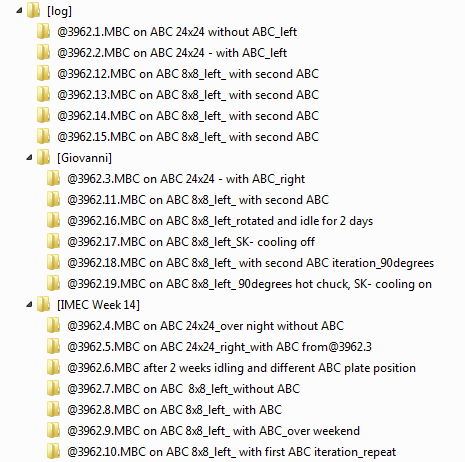
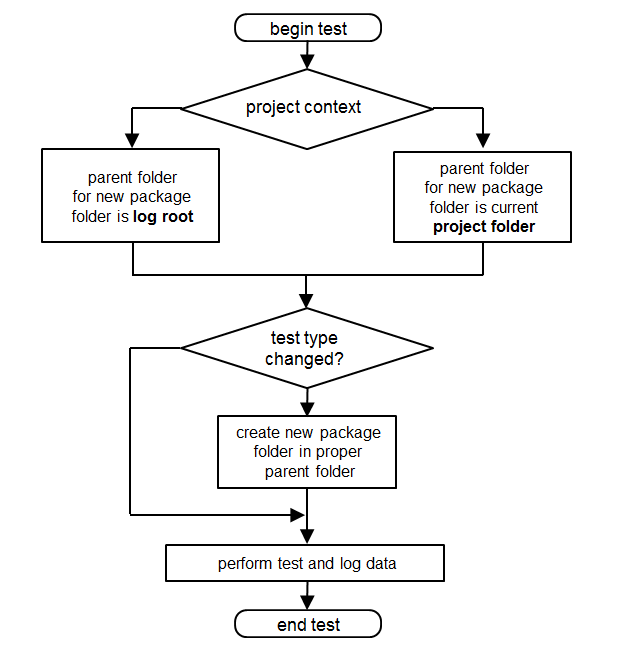


Figure 8: Example for default file organization structure

# Project Context

If a machine test is performed which writes log data to a log file package this operation can be performed in two kinds of contexts (see figure 9):

1. **Default Context**: log files are generated **without referring to a specific project**. The machine is expected **not to prompt** for additional information and writes the log files to default package folders with automatic providing of all default information.
2. **Project Context**: The user has previously created or opened a project and all log file information is expected to be saved with respect to an associated project folder. In addition the machine will prompt at begin of each test to allow the user providing a test title and commenting text, which will be included into the log file information.



no

yes

no

yes

Figure 9: Package folder management with respect to project context

# Project Context Transitions

The transition diagram for the project context with respect to user operations is shown in figure 10.

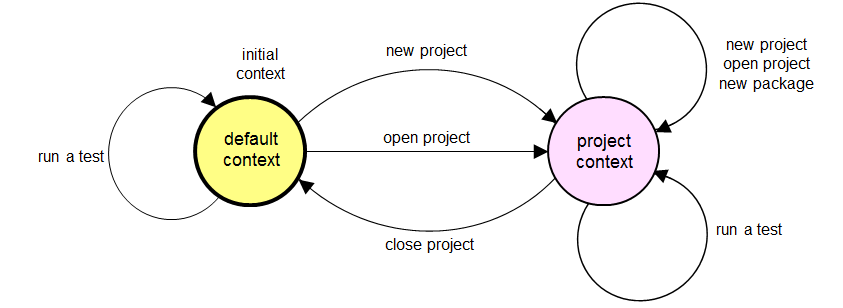


Figure 10: State diagram for project context

There are dialogs for

* creating a project (New Project)
* opening a project (Open Project)
* closing a project (Close Project)
* creating a new package

The project and package context should be remembered during machine reboots and shutdown.

# Creating a Project

To create a new project the dialog of figure 11 is activated.

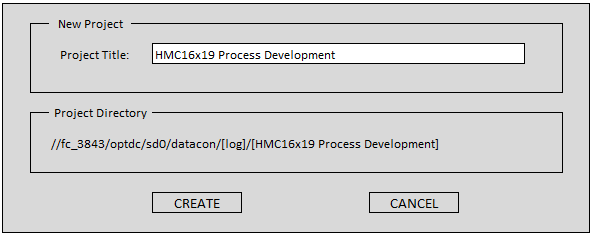


Figure 11: State diagram for project context

* The dialog asks to input a project name. Upon confirmation with **CREATE** the dialog generates the name for the project folder which is the project title enclosed by brackets (e.g. *[HMC16x19 Process Development]* for title *HMC16x19 Process Development*. Is it crucial that each project folder name begins with a bracket and ends with a bracket, as it is likewise crucial that each package folder name begins with a package identifier (@.....).
* The package folder location by the machine is always as a child folder of the log root. If the project folder exists already, the creating operation is rejected, otherwise the project folder will be created, and there is a transition of the project context state to *project context*.
* The actual test type will be reset to an undefined type.
* The current project is defined by the associated project folder, which (by dropping the brackets) allows always to re-construct the project title.

# Opening a Project

This operation opens a file selector box to select a project directory. Only folders with a name enclosed by brackets are allowed to be opened, otherwise the open operation will be rejected.

* The project context state changes to *project context*, and the open project refers to the project folder.
* The actual test type will be reset to an undefined type.
* The current project is defined by the associated project folder, which (by dropping the brackets) allows always to re-construct the project title.

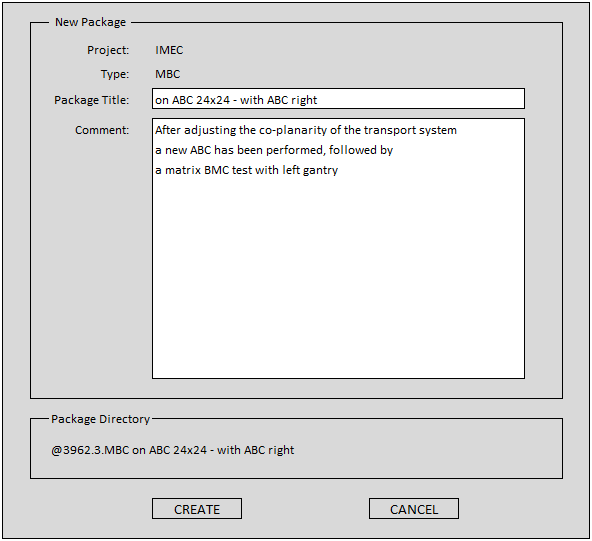
# Closing a Project

This operation closes the current active project in case of a *project context*, otherwise (*in default context*) will reject the close operation by displaying a message box to inform that there was no open project.

* The project context state changes to *default context* and there will not be a reference to a project folder any more. The actual test type will be reset to an undefined type.

# New Package Creation

A new package might be created implicitly (if the test type changes – see next section) or explicitly triggered by the user. In both cases the same dialog of figure 12 will open, and the same actions will happen. After opening of the dialog the fields for package title and comment are initially empty.



History

Figure 12: New Package dialog

* The dialog requests two text inputs, *title* and *comment,* which will be logged in the parameter section of each subsequent log file. The title can consist of any characters, not exceeding a limit of max. 40 characters. In the example of figure 12 the following parameter clauses will be subsequently logged in each log file.

$type='MBC'

$project='IMEC'

$title='on ABC 24x24 – with ABC right'

$comment='After adjusting the co-planarity of the transport system'

$comment='a new ABC has been performed, followed by'

$comment='a matrix BMC test with left gantry'

$machine='9508842003962'

$package='@3962.3'

* The comment section is an unlimited sequence of comment lines, each containing max 80 characters.
* The limitation of title and comment line length is intentionally, as the title is often displayed in diagrams and would lead to an ugly graphical layout, if it would not be limited in length. A similar rationale applies to visualization of the comments.
* If the dialog opens there is already a valid *project* and *type* context. If this condition is not valid, the dialog is not allowed to be opened. Both informations are displayed in the header of the dialog.
* The user is asked to enter a package title. Any sequence of 0-40 characters is allowed to be entered. At the bottom of the name of the package directory is being displayed. It consists of the package ID (e.g. *@3962.3*) appended by the dot separated type identifier (e.g. *.MBC*) followed by the space separated title (entered just by the user). If the title is empty the package folder name terminates with the type identifier. The whole package name might look like

*@3962.3.MBC on ABC 24x24 – with ABC right* or *@3962.3.MBC*

* The package identifier (e.g. *@3962.3*) can be generated automatically by the machine by taking the last 4 digits of the machine number, concatenated with a period symbol and a (continuously incremented) package number. The latest package number can be always memorized (or reconstructed) by the machine, and will be incremented with each creation of a new package.
* The user is also advised to enter 0 or more (unlimited) comment lines. The dialog provides a *history recall* function, comprised by a next/previous button to recall historical *title* and *comment* contents, which are pasted into the *title*/*comment* fields, which might be subsequently modified. This is an efficient function for the case where test series have similar *package titles* and *comments*, and the user is aimed to make small modifications like different parameter settings.
* Upon termination of the dialog with *CREATE* the machine checks whether it is able to create the designated package directory. If not, the dialog shows an error message and resumes the dialog until there is a valid user input or a dialog termination with *CANCEL*.

# Performing a Test

If a test is being performed further control will perform according to figure 9.

* In a first step it is being checked whether the project context state is either *default context* or *project context*.
* In case of *default context* the log root (…/[log] folder) will be considered as the *parent folder* for further package folder creation.
* In case of a *project context* the associated *project folder* will be considered as the *parent folder* for further package folder creation.
* In the next step it is checked whether the current test *type* is matching the *type* of the previously performed test. If e.g. the previously performed test was of type *BMC*, and the current test is also a *BMC* test the *test type has not changed*, but if the previously performed test was a vibration test (type *VIB*) and the current test is a *BMC* test, the *test type has changed*. Keep also in mind that after project *creation* or *open* the previous test type will be set to *undefined* and the test type may always be considered as *changed*.
* If the test type has *not changed* the machine is able to remember the previous package folder and will create the new log file in the same package folder. Note that this is consistent with the suggestion to keep only one type of log files in a package folder, and the name of the package folder tells already about this type.
* If the test type has changed it is necessary to create a new package folder as a child of the previously determined parent folder. The machine will open a *New Package* dialog (see figure 12), and the user is expected to input proper information which are required for a new package creation (see next section).
* After that the log file is created in the package folder. Note that the log file type complies always to the *type* of the package folder.

1. It should be noted that the log files which are addressed here are not refering to the machine's event log file. [↑](#footnote-ref-1)
2. There are log file formats like *8800 stage logging* or some kinds of event logging which do not fit into this standard format [↑](#footnote-ref-2)